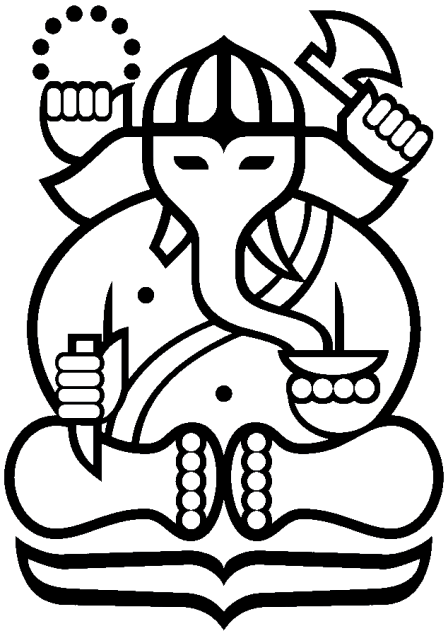
Laporan Tugas Besar 1 IF3270 Pembelajaran Mesin

Bagian A



Anggota :

13521010	Muhamad Salman Hakim Alfarisi
13521013	Eunice Sarah Siregar
13521018	Syarifa Dwi Purnamasari
13521027	Agsha Athalla Nurkareem

Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2023

Daftar Isi

Penjelasan Implementasi

Neural Network merupakan sebuah metode kecerdasan buatan yang mengajarkan komputer untuk memproses data dengan cara yang terinspirasi oleh otak manusia. Dalam *machine learning*, *neural network* mengacu pada sekumpulan algoritma yang di *design* untuk membantu mesin dalam menemukan pattern tanpa diprogram secara eksplisit. Dalam neural network terdapat tiga hal dasar, yaitu *input layer*, *hidden layer*, dan *output layer*.

Pada kali ini, dirancang algoritma untuk melakukan *feed forward* dari input yang diberikan dalam bentuk file JSON dengan memanfaatkan beberapa *library* yang terdapat pada Python, seperti numpy untuk melakukan perhitungan, json untuk load model dari file JSON, networkx dan matplotlib.pyplot untuk visualisasi *graph*. Program mengimplementasikan beberapa fungsi aktivasi, seperti linear, ReLU, sigmoid, dan softmax dengan detail formula sebagai berikut.

$$net = \Sigma x_i w_i$$

$$Linear\ f(net) = net$$

$$ReLU\ f(net) = max(0, net)$$

$$Sigmoid\ f(net) = \frac{1}{(1+e^{-net})}$$

$$Softmax\ f(net) = \frac{e^{net_i}}{\Sigma e^{net_i}}$$

Implementasi dimulai dari pendefinisian activation function yang berisi jenis activation function dan formula perhitungannya.

```
activation_functions = {
    'relu': lambda x: np.maximum(0, x),
    'sigmoid': lambda x: 1 / (1 + np.exp(-x)),
    'linear': lambda x: x,
    'softmax': lambda x: np.exp(x) / np.sum(np.exp(x))
}
```

Selanjutnya dibuat fungsi read model untuk membaca data model, layers, weights, dan input array dari json yang kemudian disimpan ke dalam array.

```
def read_model(filename):
```

```
    with open("testcase/" + filename, 'r') as file:
        data = json.load(file)
        model = data["case"]["model"]
        layers = data["case"]["model"]["layers"]
        weights = data["case"]["weights"]
        input_array = np.array(data["case"]["input"])
    return model,layers, weights, input_array
```

Untuk membuat penghitungan FFNN, dibuat sebuah kelas FFNN yang berisi fungsi dan prosedur sebagai berikut.

1. Prosedur add_layer untuk menambahkan layer yang terdiri dari neuron, function, weights, dan bias.
2. Fungsi forward untuk melakukan proses feedforward.

```
class FFNN:
    def __init__(self, input_array, layers):
        self.input_array = input_array
        self.layers = layers
        self.output = None

    def add_layer(self, neuron, function, weights, bias):
        layer = {
            'neuron': neuron,
            'function': function,
            'weights': weights,
            'bias': bias
        }
        self.layers.append(layer)

    def forward(self):
        self.output = self.input_array
        for layer in self.layers:
            weights = layer['weights']
            bias = layer['bias']
            function = activation_functions[layer['function']]
```

```
            self.output = function(np.dot(self.output, weights) + bias)
        return self.output
```

Selanjutnya terdapat prosedur visualizeGraph untuk pembuatan visualisasi dari FFNN. Prosedur ini melakukan pembacaan dari model dan weights yang diproses pada main. Proses dimulai dengan identifikasi jumlah weights kemudian pembuatan node dan edge untuk input layer, output layer, dan hidden layer (jika weights lebih dari 1). Setelah itu, dilakukan peletakan posisi node agar sesuai layer.

```
def visualizeGraph(model, weights, title):
    input_size = model['input_size']
    layer = model['layers']
    number_of_neurons = len(weights[0][0])
    G = nx.DiGraph()

    if (len(weights)==1):
        for i in range(input_size + 1):
            if (i==0):
                G.add_node(f'b{i}', layer=0)
            else:
                G.add_node(f'x{i}', layer=0)

        for i in range(number_of_neurons):
            G.add_node(f'o{i}', layer=1)

        for i in range(input_size + 1):
            for j in range(number_of_neurons):
                weight = weights[0][i][j]
                if (i==0):
                    G.add_edge(f'b{i}', f'o{j}', weight=weight)
                else:
                    G.add_edge(f'x{i}', f'o{j}', weight=weight)

    else:
        layers = 0
        for i in range(input_size + 1):
```

```
            if (i==0):
                G.add_node(f'b{i}', layer=0)
            else:
                G.add_node(f'x{i}', layer=0)

        for i in range(len(weights)-1):
            layers += 1
            G.add_node(f'b{i+1}',layer=layers)
            for j in range(len(weights[i][0])):
                G.add_node(f'h{i+1}{j+1}', layer=layers)

        layers += 1
        for i in range(len(output[0])):
            G.add_node(f'o{i}', layer=layers)

        bCount=0
        for i in range (len(weights)):
            for j in range (len(weights[i])):
                for k in range (len(weights[i][j])):
                    # print(f'i={i}, j={j}, k={k}')
                    if (j==0):
                        if (i==len(weights)-1):
                            G.add_edge(f'b{bCount}', f'o{k}',
weight=weights[i][j][k])
                        else:
                            G.add_edge(f'b{bCount}', f'h{i+1}{k+1}',
weight=weights[i][j][k])
                    else:
                        if (i==0):
                            G.add_edge(f'x{j}', f'h{i+1}{k+1}',
weight=weights[i][j][k])
                        elif (i==len(weights)-1):
                            G.add_edge(f'h{i}{j}', f'o{k}',
weight=weights[i][j][k])
                        else:
```

```
                            G.add_edge(f'h{i}{j}', f'h{i+1}{k+1}',
weight=weights[i][j][k])
            bCount+=1



    pos = {}
    y_distance = 150
    current_y = 0
    for node in G.nodes():
        layer = G.nodes[node]['layer']
        if layer not in pos:
            pos[layer] = {}
        if node.startswith('o'):
            pos[layer][node] = (layer, -len(pos[layer]) * y_distance)
        else:
            pos[layer][node] = (layer, -len(pos[layer]) * y_distance)

    final_pos = {node: pos[layer][node] if node in pos[layer] else
default_pos[node] for layer in pos for node in pos[layer]}



    # Plot the graph
    plt.figure(figsize=(20,12))
    nx.draw(G, final_pos, with_labels=True, node_size=1000,
node_color='lightblue', font_size=10, edge_color='black', linewidths=1,
arrows=True)
    edge_labels = {(i, j): f'{G[i][j]["weight"]:.2f}' for i, j in G.edges()}
    nx.draw_networkx_edge_labels(G, final_pos, edge_labels=edge_labels,
font_color='red', label_pos=0.8)
    plt.title('Feedforward Neural Network')
    plt.savefig(f'{title}.png')
    plt.show()
```

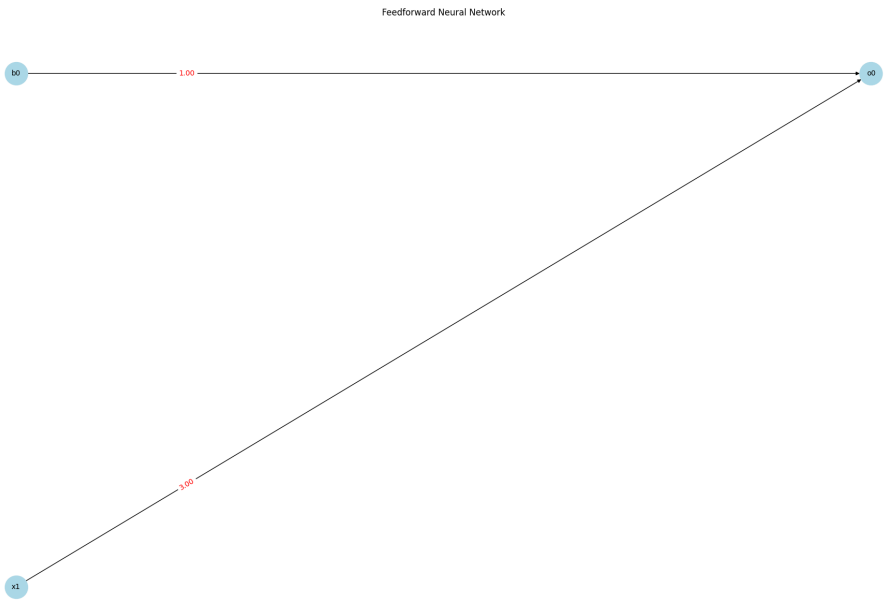
Berikut adalah main code untuk implementasi FFNN. Pertama, sistem akan meminta masukan berupa nama file yang akan dibaca dalam jenis file json. Selanjutnya, hasil pembacaan akan disimpan dalam variabel model, layers, weights, dan input_array. Array yang disimpan pada variabel input_array nantinya akan diaktivasi menggunakan masing-masing aktivasi sesuai permintaan pada file json. Setelah dilakukan aktivasi, akan dilakukan visualisasi dengan memanggil fungsi visualizeGraph.

```
model_filename = input("Input the file model name: ")
model, layers, weights, input_array = read_model(model_filename)

ffnn = FFNN(input_array, [])
for layer, weight in zip(layers, weights):
   ffnn.add_layer(layer["number_of_neurons"], layer["activation_function"],
np.array(weight[1:]), np.array(weight[0]))

output = ffnn.forward()
print("Output:", output)

visualizeGraph(model, weights, "model")
```

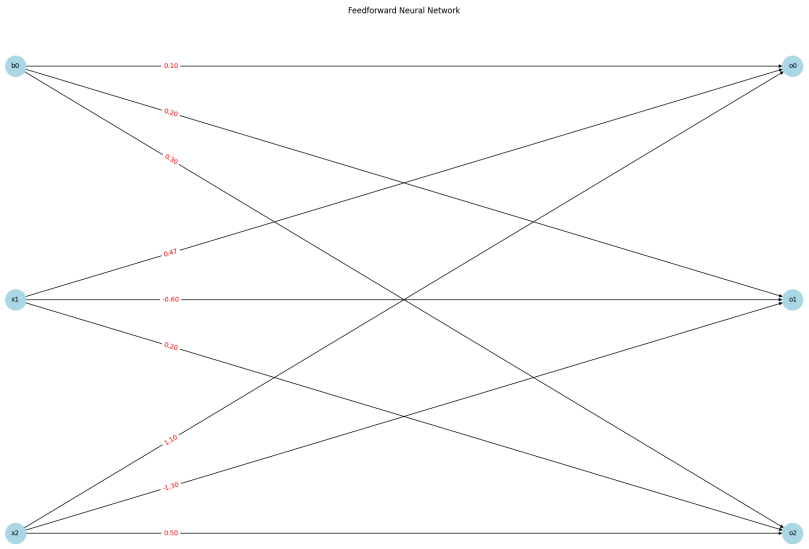
Hasil Pengujian

Setelah melakukan perancangan algoritma, dilakukan pengujian untuk setiap hasil test case yang telah diberikan. Berikut adalah hasil pengujian dari setiap test case.

1. Linear

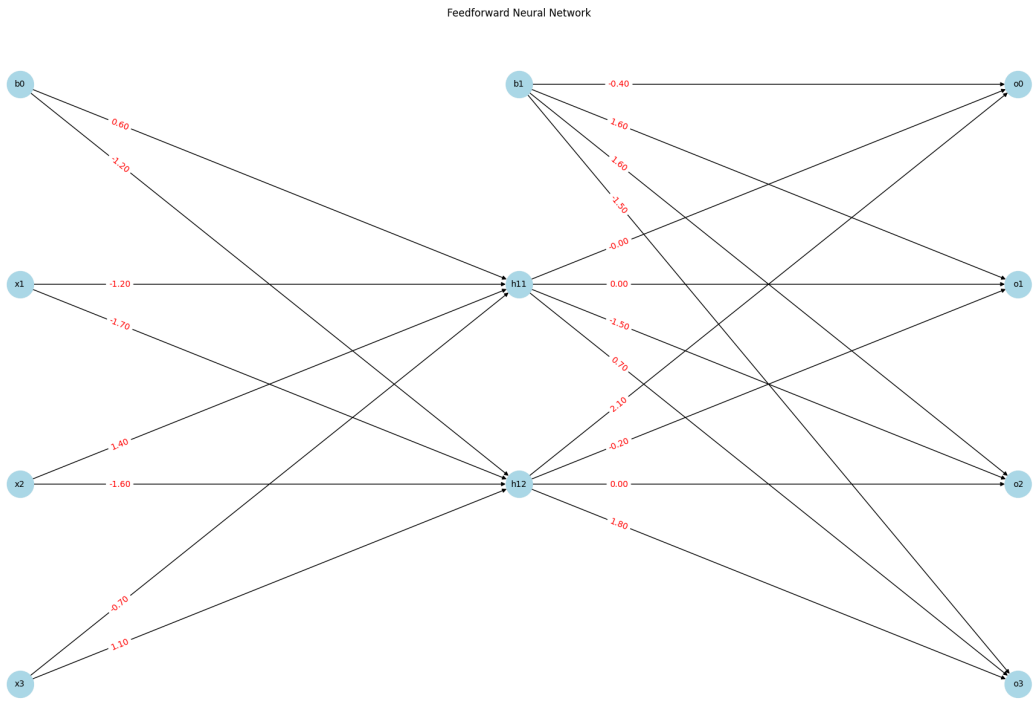
Hasil Pengujian

Output: `[[-11.] [ -8.] [ -5.] [ -2.] [  1.] [  4.] [  7.] [ 10.] [ 13.] [ 16.]]`

2. ReLU

Hasil Pengujian

`Output: [[0.31  0.    0.375]]`

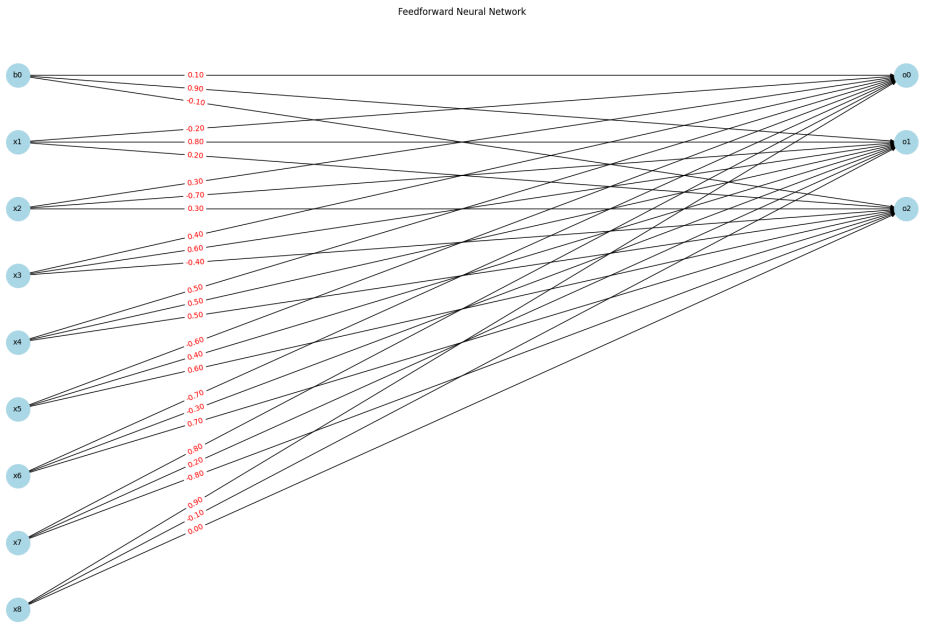
3. Sigmoid

Hasil Pengujian

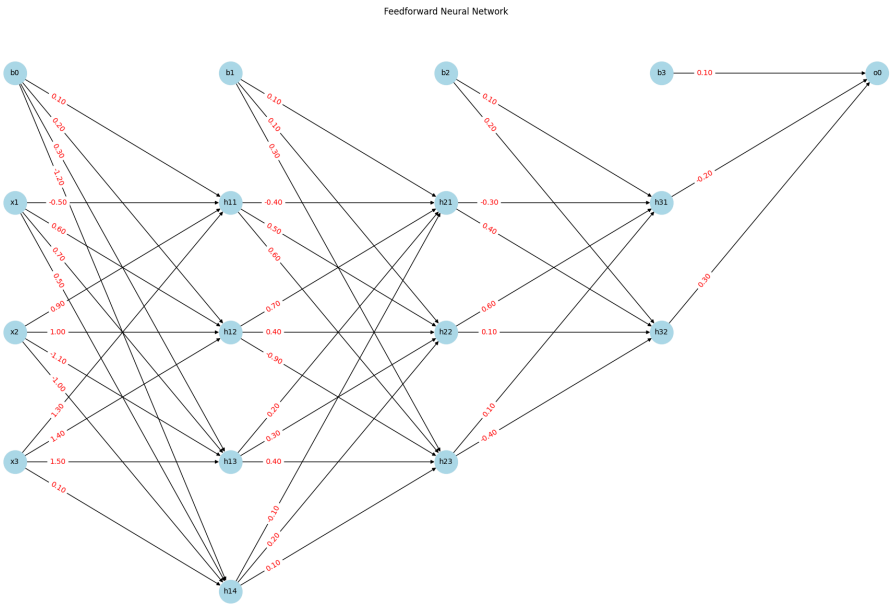



```
Output: [[0.41197346 0.8314294  0.53018536 0.31607396]
 [0.78266141 0.80843631 0.55350518 0.64278501]
 [0.58987524 0.82160954 0.75436518 0.34919895]
 [0.6722004  0.81660439 0.59020258 0.50870988]
 [0.47322841 0.82808466 0.69105452 0.29358323]]
```

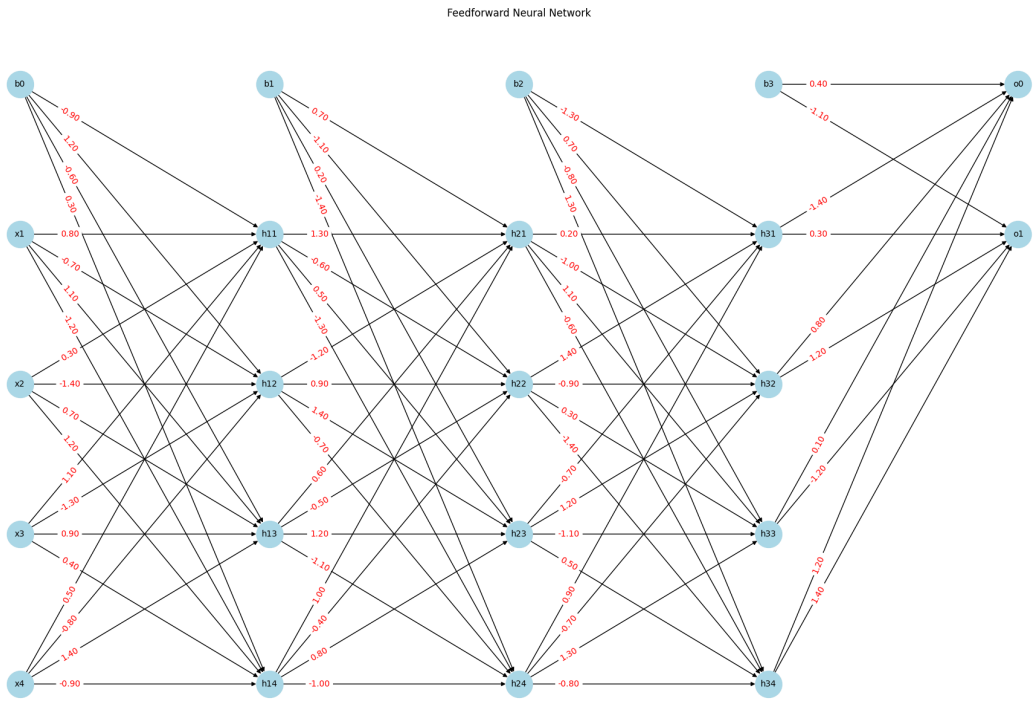
4. Softmax

Hasil Pengujian

Output: [[0.76439061 0.21168068 0.02392871]]

5. Multilayer

Hasil Pengujian

`Output: [[0.4846748]]`

6. Multilayer Softmax

Hasil Pengujian

`Output: [[0.7042294 0.2957706]]`

Perbandingan dengan Perhitungan Manual

Jika dibandingkan dengan perhitungan manual, didapatkan bahwa semua test case memiliki akhir yang sama antara test case, perhitungan manual, dan algoritma *feed forward* yang telah dirancang sebelumnya. Berikut adalah perbandingan beserta detail perhitungan manual untuk setiap test case.

1. Linear

Algoritma	Perhitungan Manual
`Output: [[-11.] [ -8.] [ -5.] [ -2.] [ 1.] [ 4.] [ 7.] [ 10.] [ 13.] [ 16.]]`	$w_0 = 1; w_1 = 3;$ $x_{i=}[-4.0], [-3.0], [-2.0], [-1.0], [0.0], [1.0], [2.0], [3.0], [4.0], [5.0]$ Formula $Output = (x_i \times w_1) + (w_0 \times 1)$ $Output_1 = (-4 \times 3) + (1 \times 1) = -11$ $Output_2 = (-3 \times 3) + (1 \times 1) = -8$ $Output_3 = (-2 \times 3) + (1 \times 1) = -5$ $\ldots dst$ $Output = -11, -8, -5, -2, 1, 4 7, 10, 13, 16$

2. ReLU

Algoritma	Perhitungan Manual
`Output: [[0.31 0. 0.375]]`	$w = [[0.1, 0.2, 0.3],$ $[0.47, -0.6, 0.2],$ $[1.1, -1.3, 0.5]]$ $x_{i=}[1.5], [-0.45]$ $Output_1 = (0.1 \times 1) + (1.5 \times 0.47) (-0.45 \times 1.1) = 0.31$ $Output_2 = (0.2 \times 1) + (1.5 \times -0.6) (-0.45 \times -1.3) = -0.115$ $Output_3 = (0.3 \times 1) + (1.5 \times 0.2)$

	$(-0.45 \times 0.5) = 0.375$ $Output_1 = max(0, Output_1) = 0.31$ $Output_2 = max(0, Output_2) = 0$ $Output_3 = max(0, Output_3) = 0.375$

3. Sigmoid

Algoritma	Perhitungan Manual

`Output: [[0.41197346 0.8314294 0.53018536 0.31607396]`
`[0.78266141 0.80843631 0.55350518 0.64278501]`
`[0.58987524 0.82160954 0.75436518 0.34919895]`
`[0.6722004 0.81660439 0.59020258 0.50870988]`
`[0.47322841 0.82808466 0.69105452 0.29358323]]`
	Hitung layer pertama dengan aktivasi Sigmoid.
	$net_{11} = (0.6 \times 1) + (-1.2 \times -0.6) + (1.4 \times 1.6) + (-0.7 \times -1) = 4.26$
	$h_{11} = 1/1 + e^{-net_{11}} = 0.9860743597$
	$\ldots dst$
	Didapatkan hasil sebagai berikut.
	(table below)
	Hitung layer output dengan aktivasi Sigmoid.
	$net_0 = (-0.4 \times 1) + (0 \times 0.9860743) + (2.1 \times 0.02104134702) = -0.3558131713$
	$0_0 = 1/1 + e^{-net_{11}} = 0.4119734556$

hix	net	f_net
h_{11}	4.26	0.9860743597
h_{12}	-3.84	0.02104134702
h_{21}	2.49	0.9234378026
h_{22}	1.39	0.8005922432
h_{31}	-0.76	0.3186462662
h_{32}	-0.56	0.3635474597
h_{41}	1.54	0.8234647252
h_{42}	0.13	0.5324543064
h_{51}	0.12	0.5299640518
h_{52}	-1.82	0.139433873

	$. . . dst$ Didapatkan hasil sebagai berikut

Output	net	f(net)
o11	-0.3558131713	0.4119734556
o12	1.595791731	0.8314293994
o13	0.1208884605	0.5301853633
o14	-0.7718735236	0.3160739649
o21	1.281243711	0.7826614091
o22	1.439881551	0.8084363083
o23	0.214843296	0.5535051761
o24	0.5874724995	0.6427850098
o31	0.3634496654	0.5898752435
o32	1.527290508	0.8216095373
o33	1.122030601	0.7543651777
o34	-0.6225621862	0.3491989467
o41	0.7181540434	0.6722003954
o42	1.493509139	0.816604391
o43	0.3648029122	0.5902025844
o44	0.03484305915	0.5087098836
o51	-0.1071888668	0.473228411
o52	1.572113225	0.8280846566
o53	0.8050539224	0.6910545249
o54	-0.8780441924	0.2935832342

4. Softmax

Algoritma	Perhitungan Manual
`Output: [[0.76439061 0.21168068 0.02392871]]`	$net_0 = (0.1 \times 1) + (-0.2 \times -1) + \ldots + (0.8 \times 0.15) + (0.9 \times 0.2) = 3.022$ $net_1 = (0.9 \times 1) + (-0.8 \times -1) + \ldots + (0.2 \times 0.15) + (-0.1 \times 0.2) = 1.738$ $net_2 = (-0.1 \times 1) + (0.2 \times -1) + \ldots + (-0.8 \times 0.15) + (0 \times 0.2) = -0.442$ Selanjutnya, dilakukan aktivasi softmax dan didapatkan hasil sebagai berikut $output_0 = \frac{3.022}{(e^{3.022} + e^{1.738} + e^{-0.442})} = 0.764$ $output_1 = \frac{1.738}{(e^{3.022} + e^{1.738} + e^{-0.442})} = 0.211$ $output_2 = \frac{-0.442}{(e^{3.022} + e^{1.738} + e^{-0.442})} = 0.023$

5. Multilayer

Algoritma	Perhitungan Manual
`Output: [[0.4846748]]`	Hitung layer pertama dengan aktivasi ReLU $net_{11} = (0.1 \times 1) + (-0.5 \times -1) + (0.9 \times 0.5) + (1.3 \times 0.8) = 2.09$ $h_{11} = max(0, 2.09) = 2.09$ $\ldots dst$ Didapatkan hasil sebagai berikut. h11 2.09 h12 1.22 h13 0.25 h14 0 Hitung layer kedua dengan aktivasi ReLU $net_{21} = (0.1 \times 1) + (-0.4 \times 2.09)$

	$+ (0.7 \times 1.22) + (0.2 \times 0.25)$ $+ (-0.1 \times 0) = 0.168$ $h_{21} = max(0, 0.168) = 0.168$ $\ldots dst$ Didapatkan hasil sebagai berikut. h21 0.168 h22 1.708 h23 0.556 Hitung layer ketiga dengan aktivasi ReLU $net_{31} = (0.1 \times 1) + (-0.3 \times 0.168)$ $+ (0.6 \times 1.708)$ $+ (0.1 \times 0.556) = 1.13$ $h_{31} = max(0, 1.13) = 1.13$ $\ldots dst$ Didapatkan hasil sebagai berikut. h31 1.13 h32 0.2156 Hitung layer output dengan aktivasi Sigmoid $net_{11} = (0.1 \times 1) + (-0.2 \times 1.13)$ $+ (0.3 \times 0.2516)$ $= -0.06132$ $O_{11} = 1 / 1 + e^{-net_{11}} = 0.4846748018$

6. Multilayer Softmax

Algoritma	Perhitungan Manual
`Output: [[0.7042294 0.2957706]]`	Hitung layer pertama dengan aktivasi ReLU $net_{11} = (-0.9 \times 1) + (0.8 \times 0.1) + \ldots +$ (0.5×1.2) $= 0.64$ $\ldots dst$ $h_{11} = max(0, 0.64) = 0.64$ $h_{12} = max(0, 0.01) = 0$ $h_{13} = max(0, 1.53) = 1.53$

$h_{14} = max(0, \ -1.46) \ = 0$

Lalu, menghitung aktivasi layer selanjutnya dengan ReLU

$net_{21} = (0.7 \times 1) + (1.3 \times 0.64) + \ldots + (1 \times 0)$
$= 2.45$

$\ldots dst$

$h_{21} = max(0, \ 2.45) \ = \ 2.45$
$h_{22} = max(0, \ -2.249) \ = \ 0$
$h_{23} = max(0, \ 2.356) \ = \ 2.356$
$h_{24} = max(0, \ -3.915) \ = \ 0$

Lalu, menghitung aktivasi layer selanjutnya dengan ReLU

$net_{31} = (-1.3 \times 1) + (0.2 \times 2.45) + \ldots + (0.9 \times 0)$
$= -2.4592$

$\ldots dst$

$h_{31} = max(0, -2.4592) \ = 0$
$h_{32} = max(0, \ 1.0772) \ = 1.0772$
$h_{33} = max(0, -0.6966) \ = 0$
$h_{34} = max(0, \ 1.008) \ = 1.008$

Terakhir, hitung aktivasi layer keempat dengan aktivasi softmax

$net_{41} = (0.4 \times 1) + (-1.4 \times 0) + \ldots + (1.2 \times 1.008)$
$= 2.47$

$net_{42} = (-1.1 \times 1) + (0.3 \times 0) + \ldots + (1.4 \times 1.008)$
$= 1.60384$

Didapatkan nilai output akhir sebagai berikut

$o_0 = \frac{2.47136}{(e^{2.47136} + e^{1.60384})} \ = \ 0.704122$
$o_1 = \frac{1.60384}{(e^{2.47136} + e^{1.60384})} \ = \ 2.957706$

Pembagian Tugas

NIM	Nama	Tugas
13521010	Muhamad Salman Hakim	Membuat laporan
13521013	Eunice Sarah Siregar	Membuat laporan
13521018	Syarifa Dwi Purnamasari	Membuat algoritma FFNN
13521027	Agsha Athalla Nurkareem	Membuat visualisasi